



Chapter 1

Introduction

Da-Wei Chang

CSIE.NCKU

Source: Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, "Operating System Concepts", 10th Edition, Wiley.

Objectives



- Introduce the major **operating system components** and **functions**
- To provide coverage of basic **computer system organization**
- Many of the topics will be discussed in detail in the following chapters

Outline

- What Is an Operating System
- Computer-System Organization/Architecture
- Operating-System Structure/Operations
- Process Management
- Memory Management
- Storage Management
- Protection and Security
- Virtualization
- Basic Kernel Data Structures
- Computing Environments
- Open-Source Operating Systems

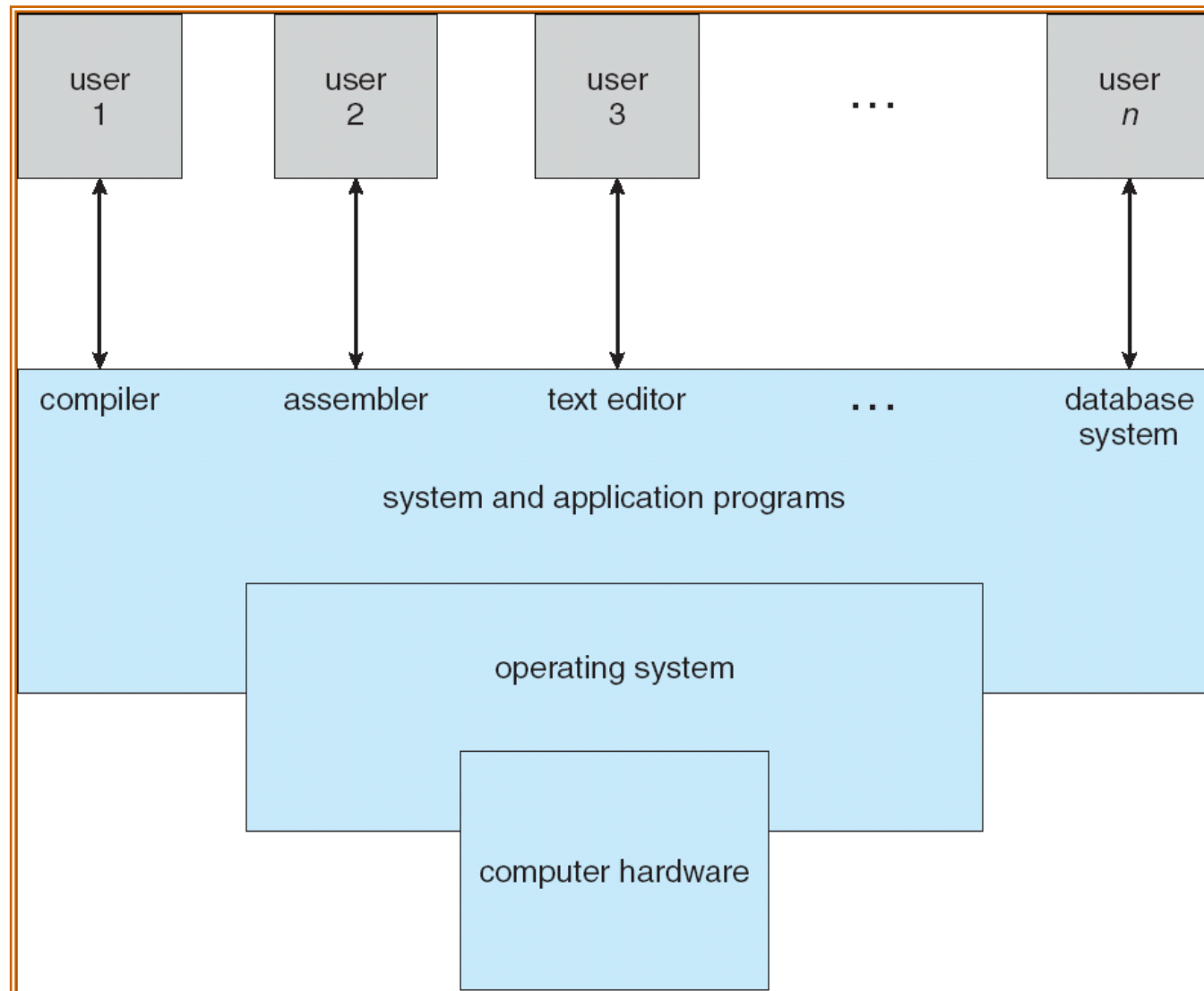
What is an Operating System?

- A **program** that acts as an intermediary between **user programs/applications** and the **computer hardware**
- Operating system goals
 - Execute user programs
 - Make the computer system convenient to use
 - Applications do not have to deal with HW directly
 - OS ensures **efficient** resource **sharing**

Computer System Structure

- A computer system can be divided into 4 components (from bottom to top)
 - **Hardware** – provides basic computing resources
 - CPU, memory, I/O devices
 - **Operating system**
 - Controls and coordinates use of hardware among various applications and users
 - **System and application programs** – solve the computing problems of the users
 - Word processors, compilers, web browsers, database systems, video games
 - **Users**
 - People, machines, other computers

Four Components of a Computer System



Operating System Definition

- **What is an operating system?**
- No universally accepted definitions
- “Everything a vendor ships when you order an operating system” is good approximation
 - But varies widely
- “The one program running at all times on the computer” (i.e., the **kernel**) Everything else is either a **system program** (ships with the operating system) or an **application program**

Operating System Definition

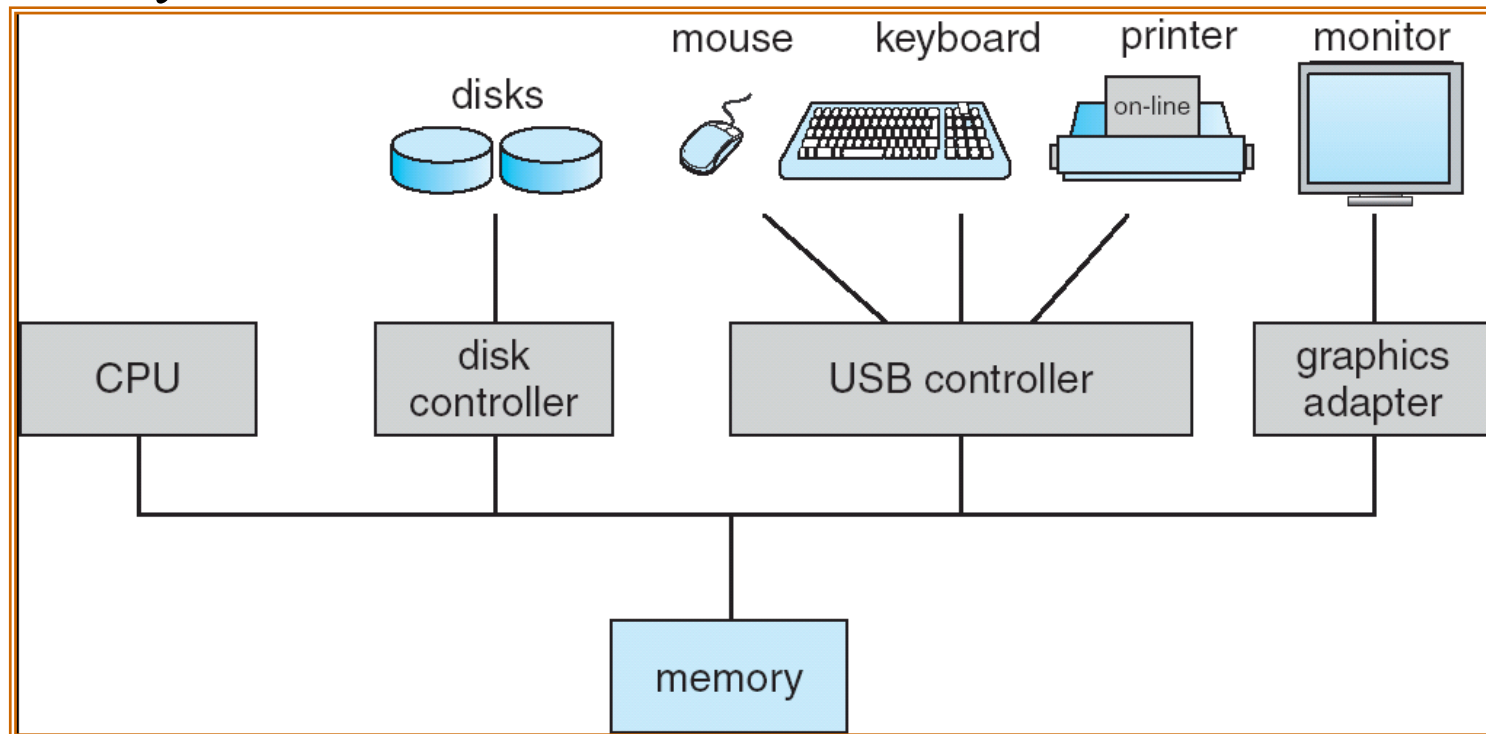
- Two points of view (in the text book)
 - OS is a **resource allocator**
 - Manages all resources
 - Applications do not have to manage resources directly
 - OS hides HW interface
 - Decides between conflicting requests for **efficient** and **fair** resource use
 - OS is a **control program**
 - Controls execution of programs to prevent **errors** and **improper use** of the computer
- Other definitions
 - A program that can monitor the other programs
 - A program that allows the other programs to **share** resources in a **controlled** manner
 - CPU, memory, IO devices....

Computer Startup

- **Who loads the OS ?**
- **Bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EEPROM, generally known as **firmware**
 - Jobs
 - Initializes all aspects of system
 - Loads operating system kernel and starts OS execution
- A bootstrap program is also called a **boot loader**
- Who loads the boot loader?
 - In PC, BIOS loads the (in-disk) boot loader
 - In many other platforms, boot loader is placed at a predefined memory address (in ROM or EEPROM)
 - Hardware jumps to the boot loader directly

Computer System Organization

- Computer-system operation
 - One or more CPUs, device controllers connect through common bus providing access to shared memory
 - Concurrent execution of CPUs and devices competing for bus cycles



Computer-System Operation

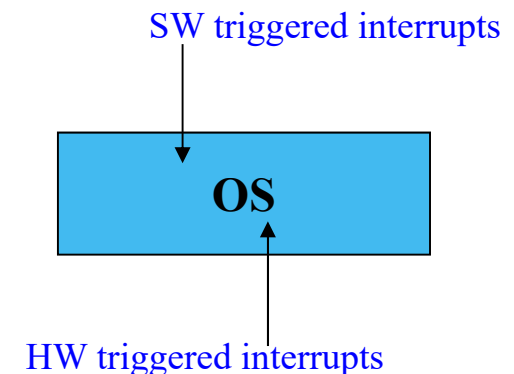
- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer.
- For data writes
 - (Device driver ← a part of OS running on the) CPU moves data from main memory to local buffers
 - Device controller writes the data from the local buffer to the device
 - Device controller informs CPU that it has finished its operation by generating an *interrupt*
- One of the jobs of an operating system is to **manage interrupts**.

Common Functions of Interrupts

- Interrupt transfers control to the **interrupt service routine (ISR)**, generally through the *interrupt vector (table)*
 - *interrupt vector* contains the **addresses or instructions** of all the service routines
 - X86: addresses
 - ARM: instructions
- The processor must save the address of the **interrupted** instruction before jumping to the ISR
- Generally, interrupts are raised by IO device controllers
 - **Hardware** interrupts

Common Functions of Interrupts

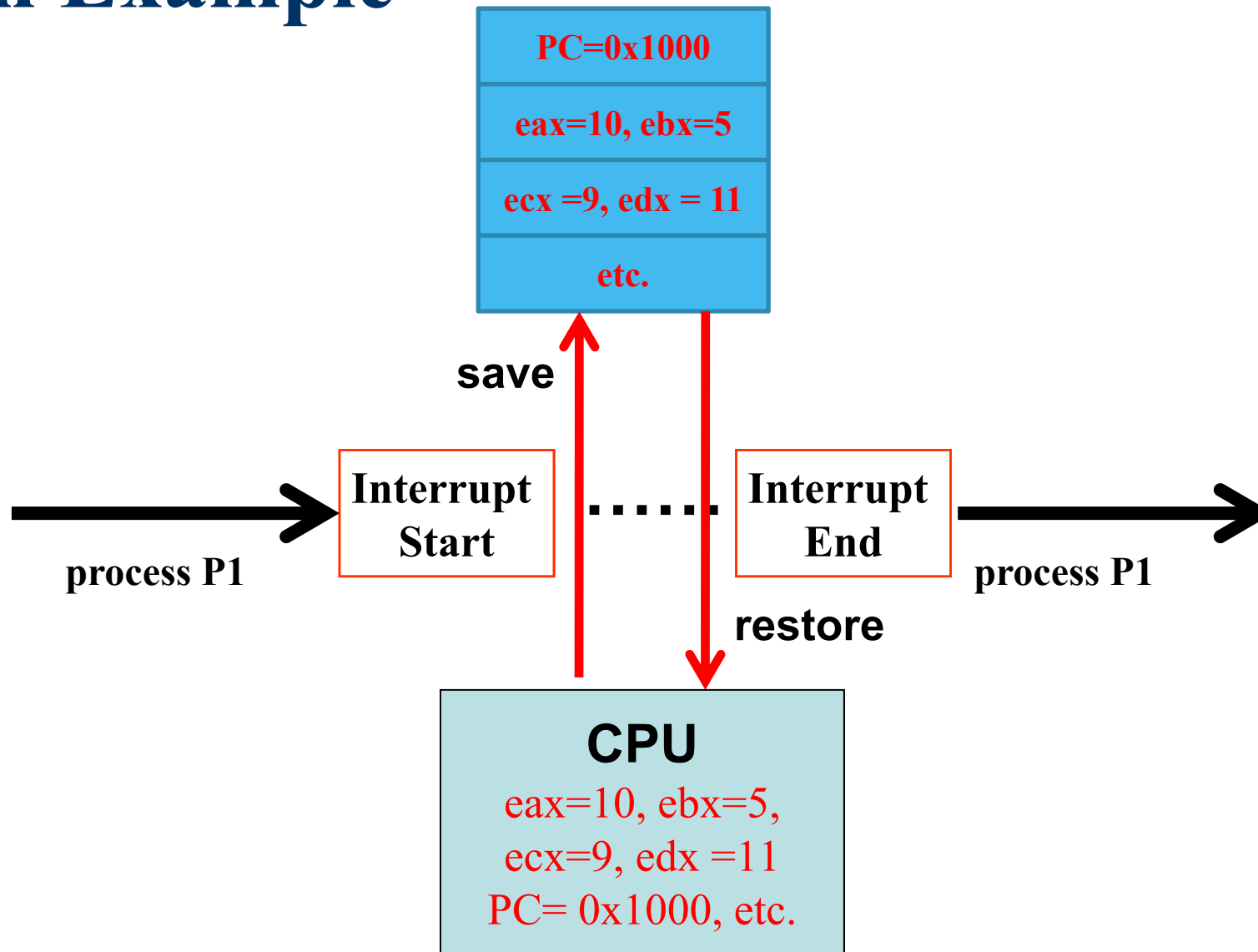
- However, *software* can also generate/raise interrupts
 - Software interrupts (or *traps*)
- A *trap* is a software-generated interrupt caused either by an *error* or a *user request*
 - Error
 - Division by zero, invalid memory access...
 - User request
 - Requests for OS services
 - *int* instruction in x86
 - *svc/swi* instruction in ARM
- An operating system is *interrupt driven*!



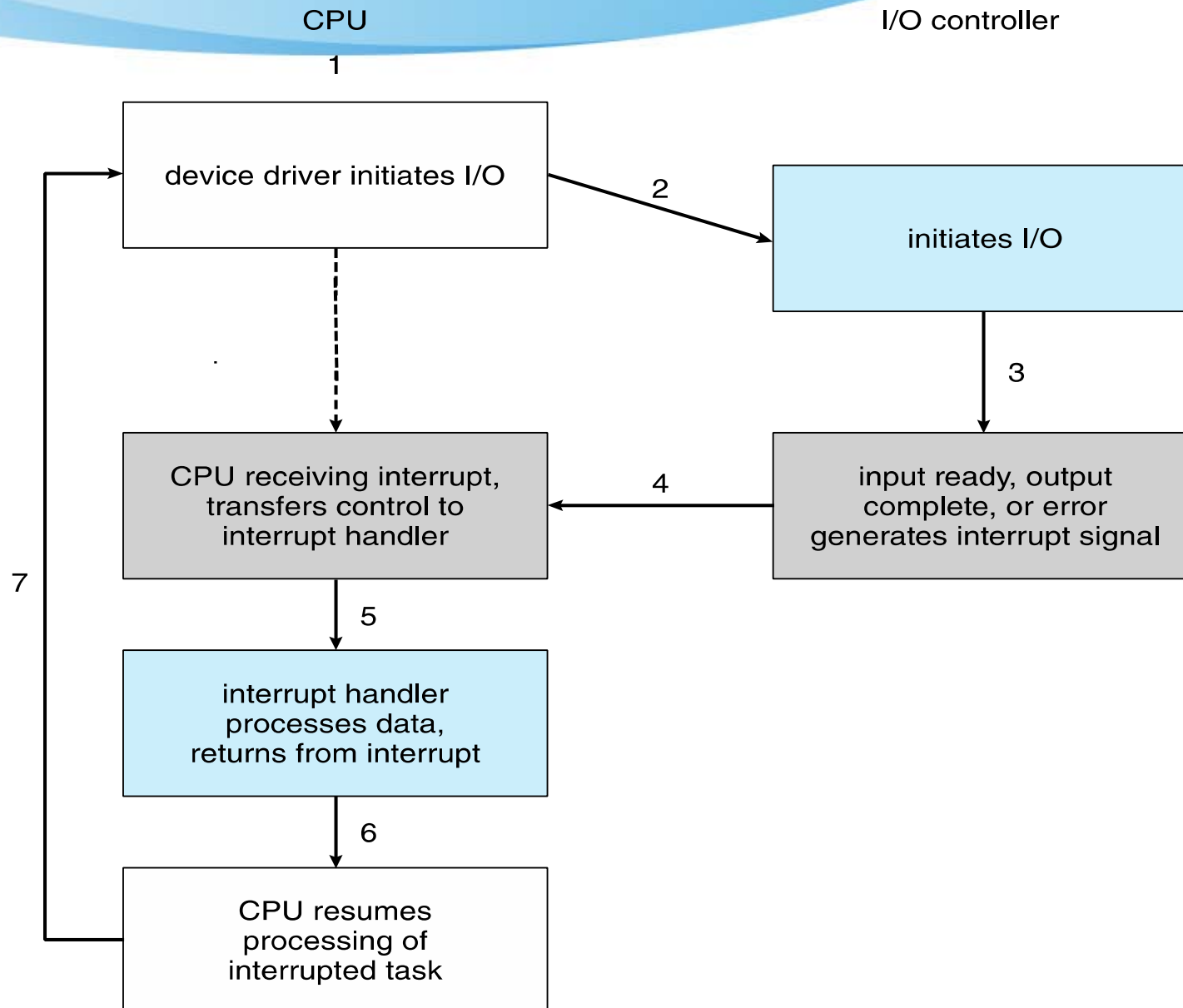
Interrupt Handling

- Save the CPU state (*registers*, including program counter)
 - Done by HW or OS
- Call the corresponding Interrupt Service Routine (ISR) or interrupt handler
 - How do you know the address of the ISR?
 - Interrupt vector table (IVT)
- Execute the ISR
- Continue the interrupted work by restoring the CPU state

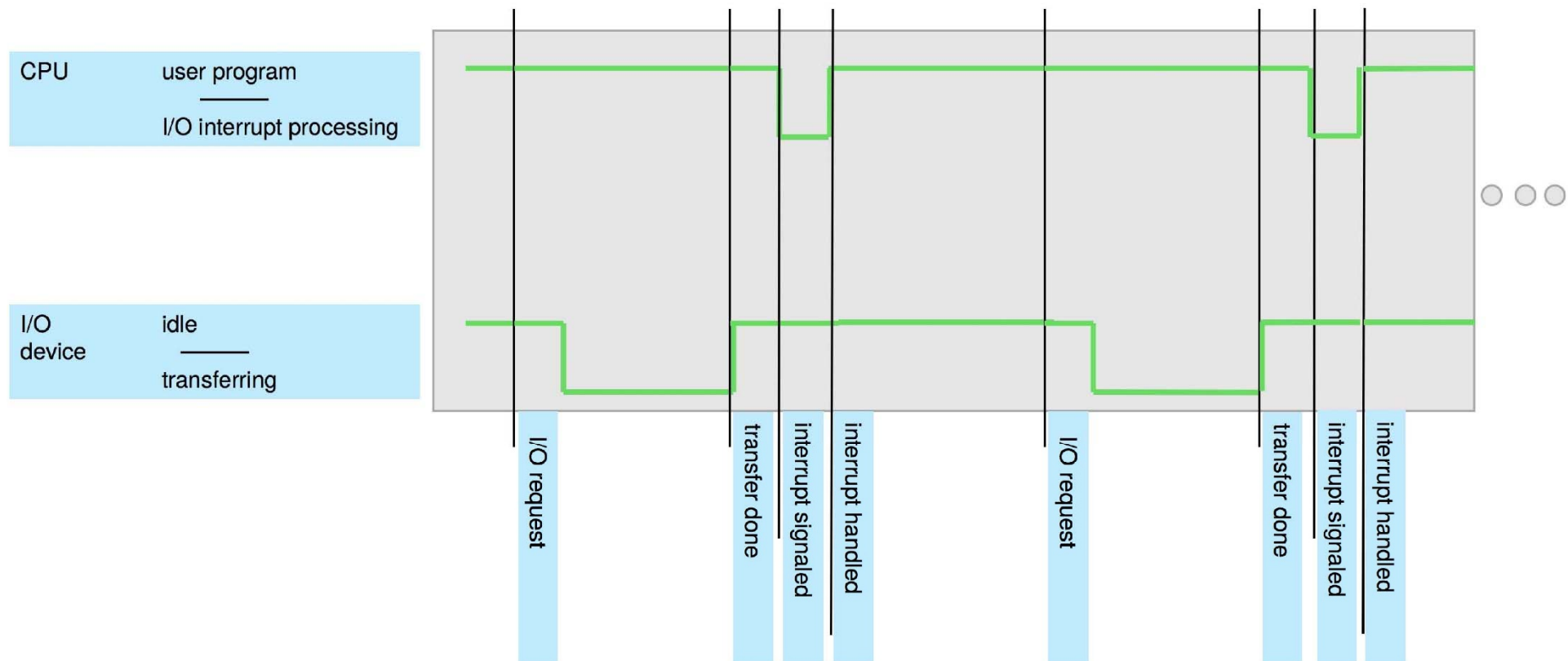
Saving/Restoring CPU State – An Example



Interrupt-drive I/O Cycle



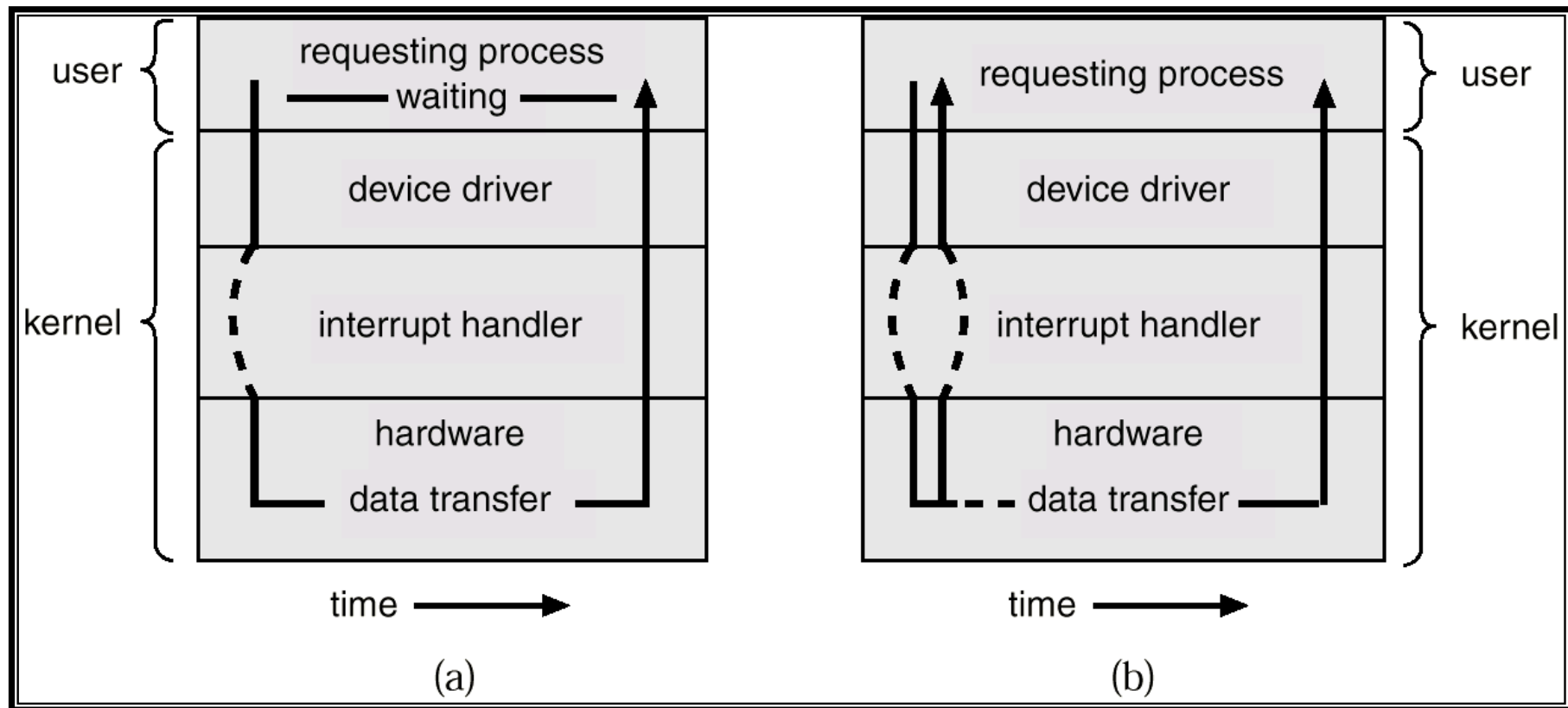
Interrupt Timeline



Two I/O Methods

Synchronous I/O

Asynchronous I/O

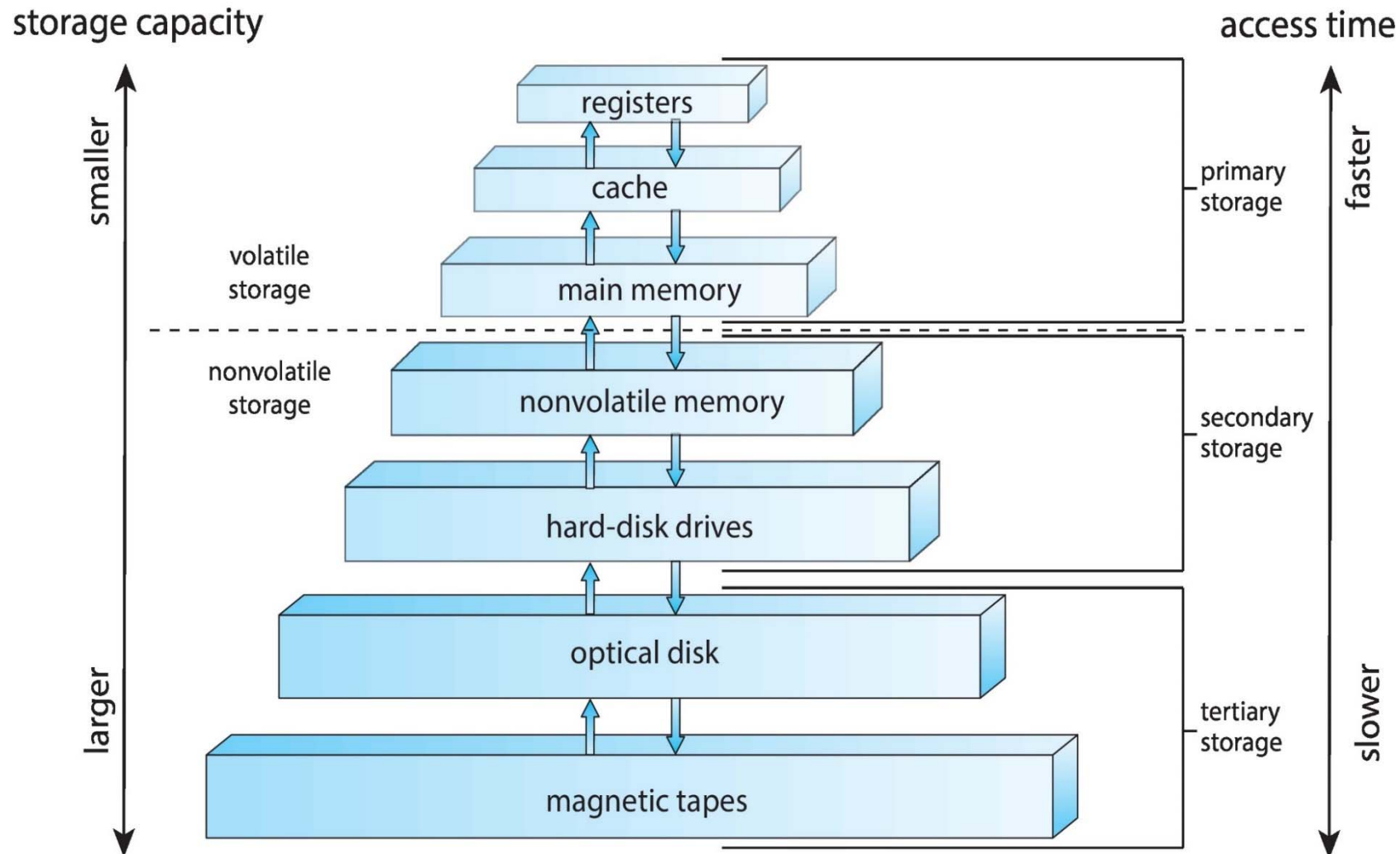


Direct Memory Access (DMA)

- Who moves the data between memory and device buffer?
 - CPU? device controller?
- DMA
 - Device controller transfers blocks of data from local buffer directly to main memory **without CPU intervention**
 - Raise interrupt when DMA transfer is done

Storage Structure

- Storage systems organized in **hierarchy**
 - The layers differ in **Sizes, Speed, Cost, Volatility...**



Storage Structure

- Primary Storage
 - Main memory –the only large storage media that the CPU can access **directly** (e.g., via load/store instructions)
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
 - Magnetic disks (HDD)
 - metal or glass platters covered with magnetic recording material
 - disk surface is logically divided into **tracks/sectors**
 - the most common secondary-storage device
 - Nonvolatile Memory (NVM)
 - SSD (Solid State Drive)
 - Becoming more popular as capacity increases and price drops
 - *Various technologies: **flash memory**, PCM, RRAM, STT-MRAM*
- Tertiary storage
 - Usually for storing backup copies

Caching

- *Caching* – information in use copied from slower to faster storage temporarily
 - main memory can be viewed as a *cache* for secondary storage
- Faster storage (cache) checked first to determine if information is there
 - If it is, information used directly from the cache (fast)
 - If not, data copied to cache and used there
- Cache is smaller than the storage being cached
 - Cache management is an important design problem
 - Cache size and replacement policy
- Performed at **many levels** in a computer system (in hardware, operating system, software)

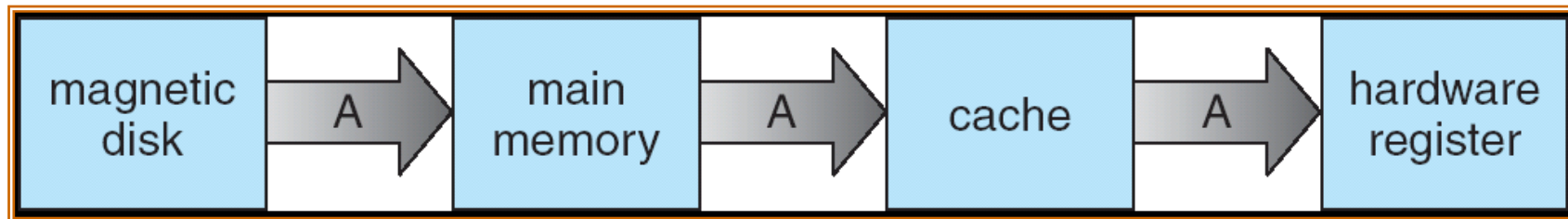
Performance of Various Levels of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	DRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be **explicit** or **implicit** (to software)

- SW-managed caching (explicit) : register, memory, disk
- HW-managed caching (implicit): cache

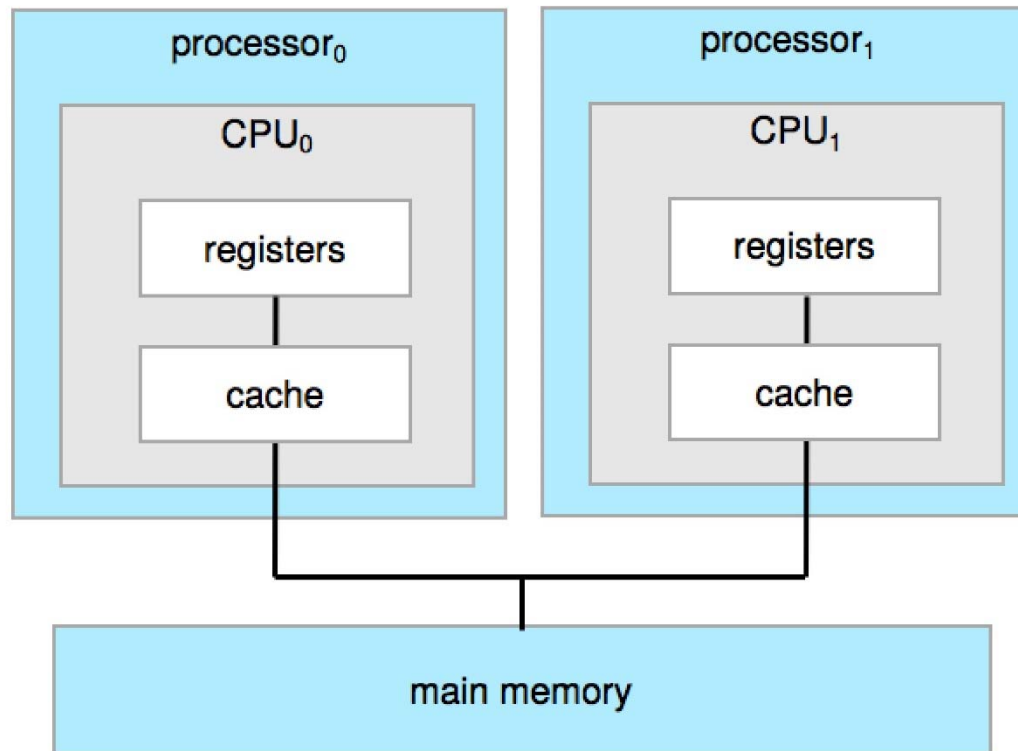
Migration of Data A from Disk to Register



Computer-System Architecture

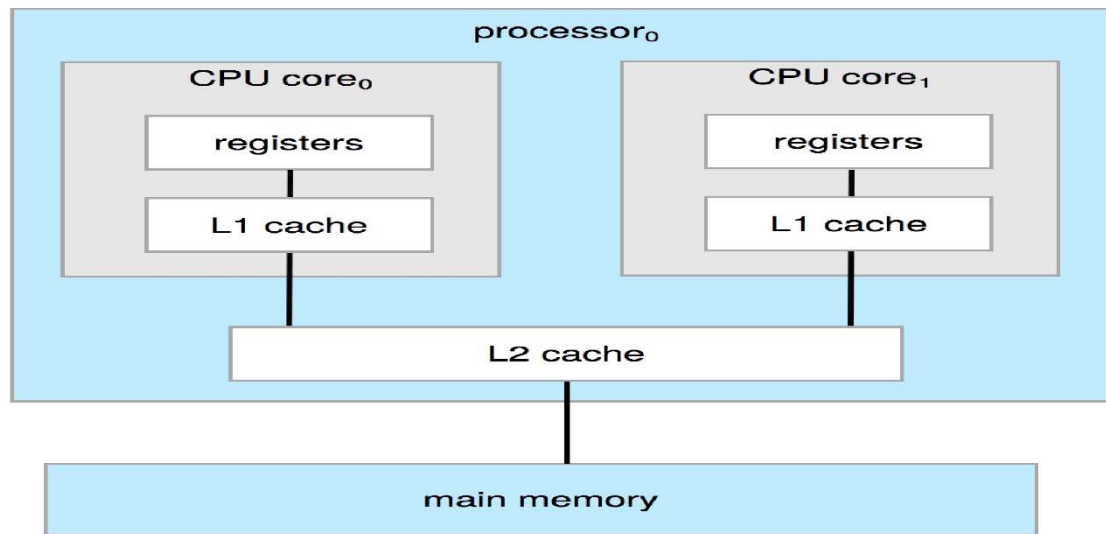
- Many systems use **general-purpose processors** (from PCs to mainframes)
 - Some systems have **special-purpose processors** as well
- **Multi-processor systems** growing in use and importance
 - Also known as **parallel systems**, **tightly-coupled systems**
 - Advantages
 1. **Increased throughput**
 2. **Increased reliability** – graceful degradation or fault tolerance
 3. **Economy of scale**
 - Multicore, cheaper than multiple single-processors
 - Types
 1. **Asymmetric Multiprocessing** (e.g. Qualcomm Snapdragon, MTK Helio)
 2. **Symmetric Multiprocessing** (e.g. Intel i7 based PC)

Symmetric Multiprocessing Architecture



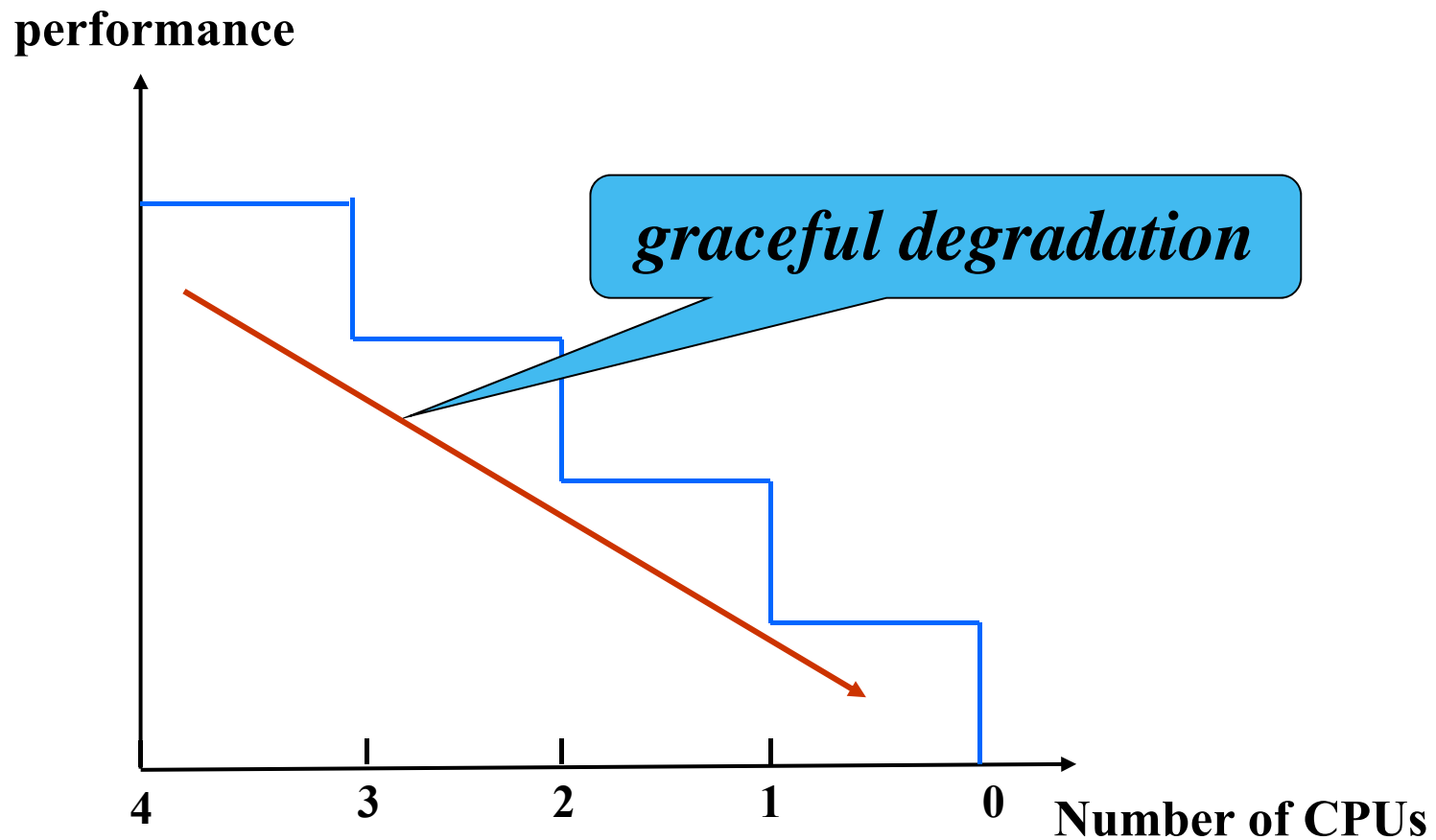
Multi-core Systems

- **Multi-core**: multiple *compute cores* on a single chip
 - More efficient than multiple single-core chips
 - On-chip communication is faster than cross-chip communication



A dual-core design with two cores on the same chip

Graceful Degradation

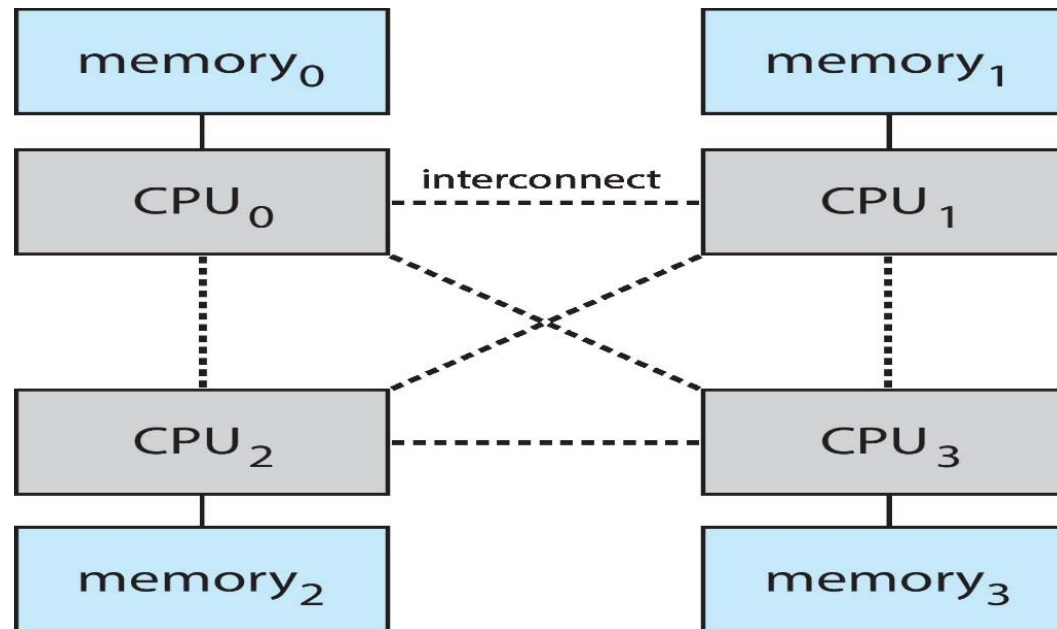


suppose a four-core system

Non-Uniform Memory Access (NUMA)

- Adding too many CPUs on multiprocessor systems will cause
 - contention for **system bus**
 - Since each CPU needs to access main memory
 - Performance begins to degrade
- Alternative approach: **NUMA**
 - Provide each CPU with its own local memory
 - Accessed via a local bus (next slide)
 - Increasingly popular on servers and high-end computing systems

Non-Uniform Memory Access (NUMA)



- Problem: memory *access time* is not the same
 - Accessing remote memory >> accessing local memory
 - OS need to manage tasks/data carefully
 - Co-locates a task and its data

Clustered Systems

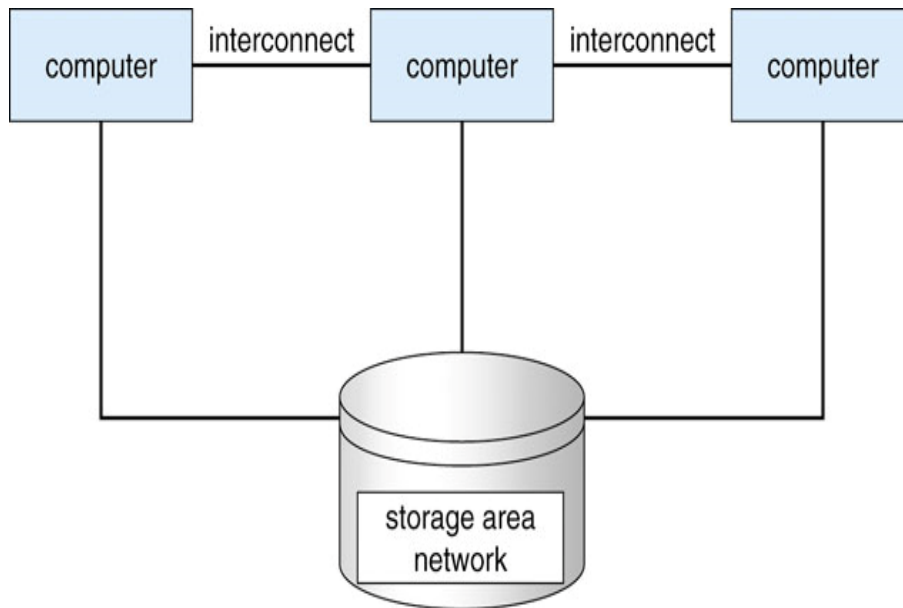
- A *clustered system*
 - Multiple computers connected via a *local area network (LAN)* or a *faster interconnection network*.
- The key of a clustered system is *high availability*
 - *Fault tolerant*: suffer a failure of any single component and continue operation
 - *Graceful degradation*: continue providing service *proportional* to the level of surviving hardware

General Structure of a Clustered System

An example

*HP C7000 G3 BladeSystem

-15 nodes, 320 cores, 10 Gb Ethernet



<https://ctoservers.co.uk/hp-c7000-g3-baldesystem-w15-x-bl460c-g8--320-cpu-cores1tb-ram-10gbe---cad-cfd--lsdyna--ansys-2101-p.asp>

Clustered Systems

- *Asymmetric Clustering*

- Some machines are in *hot-standby mode* while the others are running applications.
- A hot-standby machine (i.e., does nothing but) monitors the other machines and becomes active if one server fails.

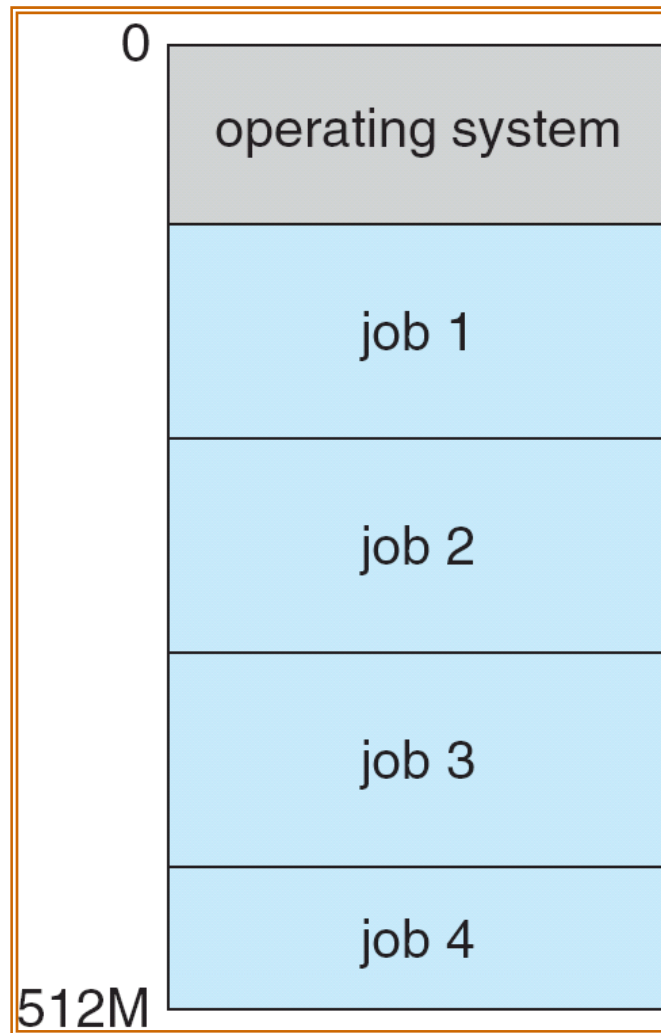
- *Symmetric Clustering*

- Machines run applications while monitoring each other
 - more efficient as it uses all available hardware

Multiprogramming & Timesharing

- **Multiprogramming** is needed for **efficiency**
 - a single job cannot keep CPU and I/O devices busy at all times
 - E.g., CPU idle if the job is waiting for IO
 - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
 - **increase CPU utilization**
 - A subset of total jobs in system is kept in memory
 - When one in-memory job has to wait (for I/O for example), OS switches to another job in the memory
 - Other jobs are kept in the **on-disk job pool**
 - A job is selected and put in memory via **job scheduling**

Memory Layout for a Multiprogrammed System



Multiprogramming & Timesharing

- **Timesharing (multitasking)** is logical **extension** to multiprogramming in which CPU switches jobs so **frequently** that users can interact with each job while it is running, creating **interactive** computing
 - **Response time** should be < 1 second
 - If several jobs ready to run at the same time \Rightarrow **CPU scheduling**
 - differences with job scheduling?
 - If processes don't fit in memory, **swapping** moves them in and out to run
 - **Virtual memory** allows execution of processes not completely in memory
 - Allows a program that is larger than the physical memory to run

Operating System Operations

- An OS must prevent user jobs/processes from damaging the normal operations of the OS
- **Dual-mode operation** allows OS to protect itself
 - **Processor** provides **user mode** and **kernel mode**
 - User processes run in user mode (non-privileged mode)
 - OS runs in kernel mode (privileged mode)
 - Some instructions designated as **privileged**, only executable in kernel mode
 - E.g. I/O control, timer, special registers, interrupt related instructions...
 - a **mode bit** is provided
 - to distinguish when system is running user code or kernel code
 - A **system call** changes the mode to the **kernel mode**
 - return from the system call resets the mode to **user mode**

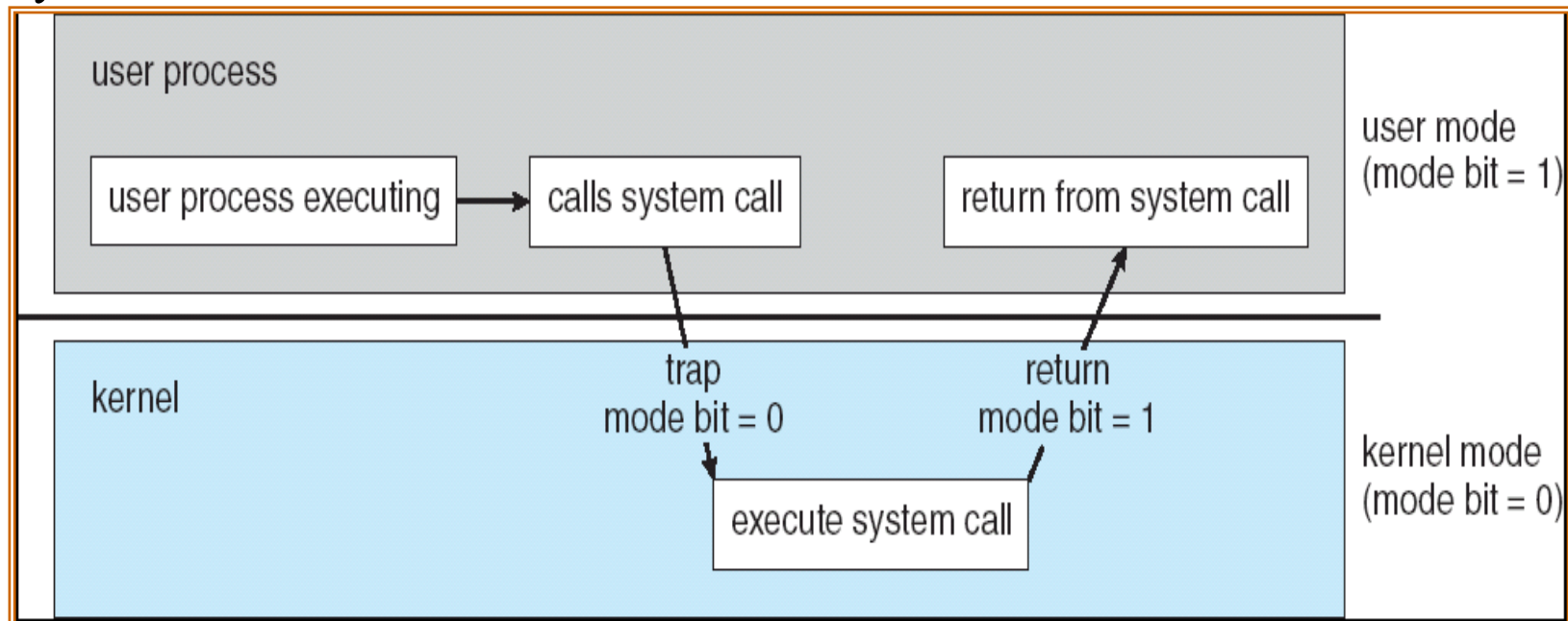
Transition from User to Kernel Mode

At system boot time, the HW starts in the kernel mode.

- after system initialization, the OS run applications in user mode.

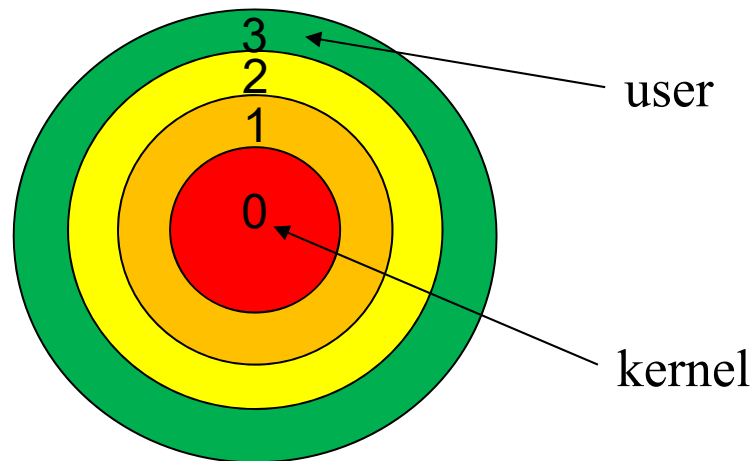
When a user process needs an OS service, it issues an **system call**
e.g., read a file, send a packet...

System call flow---



Transition from User to Kernel Mode

- Modes are provided by HW (CPU)
 - Intel's 8088 CPU has only a single mode
 - The OS (e.g., MS-DOS) can be crashed by a user program
 - Modern x86 has (more than) 4 modes/rings



Timers

- Conceptually, user processes run **on top of** the OS
 - *In fact, they run on the CPU directly!!!*
- How to ensure that the OS maintains control over the CPU?
- Prevent an infinite loop in a process from hogging resources
 - By setting up a timeout period for the process
 - Done by **timers**
 - Operating system decrements the time count for the process periodically (whenever a **timer interrupt** is generated)
 - When the count becomes zero (i.e., the current process has used up its time period)
 - hand the CPU to another process

Process Management

- A process is a program **in execution**. It is a unit of work within the system.
 - Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
- Process termination requires reclaiming reusable resources
- **Single-threaded** process has **one program counter** specifying location of **next instruction** to execute
 - Process executes instructions sequentially, one at a time, until completion
- **Multi-threaded** process has multiple program counters
 - One program counter per thread
- Typically, a system has many processes running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes/threads

Process Management Activities



- Activities related to process management
 - Creating and deleting processes
 - Suspending and resuming processes
 - Providing mechanisms for process synchronization
 - Providing mechanisms for process communication
 - Providing mechanisms for deadlock handling
- We will discuss the above topics later

Memory Management

- **Data** have to be put in memory before being processed
- **Instructions** have to be put in memory before being executed
- Memory management determines **what is in memory**
 - Goal: optimizing CPU utilization and computer's response to users
- Memory management activities
 - Allocating and deallocating memory space as needed
 - Keeping track of **which parts** of memory are currently being used and **by whom**
 - Deciding **which processes** (or parts thereof) **and data** to move into and out of memory

Storage Management

- OS provides uniform, logical view of information storage
 - Different types of storage devices (e.g., solid-state drives, hard disk drives, tape drives...)
 - **Varying properties** include access speed, capacity, data-transfer rate, access method (sequential or random)
 - **Abstracts physical properties** to **logical storage unit**
- **file**
 - OS stores files into the storage devices

Storage Management

- **File-System Management** (provide file interface)
 - File content is determined by its creator
 - Free format – e.g. text files...
 - Fixed format – e.g. executable files...
 - Files usually organized into **directories**
 - **Access control** on most systems to determine **who** can access **what**
 - OS activities include
 - Creating and deleting files and directories
 - Primitives to manipulate files and dirs
 - read/write/append files, set/get file status, set/get file permissions

Storage Management

- **Mass-Storage Management** (focus on efficient operation)
 - Usually, disks
 - Mass storages are used to store
 - data that do not fit in main memory, or
 - data that must be kept for a “long” period of time
 - OS activities
 - Free-space management (bitmap? free list?)
 - Storage allocation
 - Disk scheduling
 - Critical to system performance
 - Because disk is usually the performance bottleneck
 - Latency of a computer operation usually depends on disk subsystem and its algorithms

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem
 - Drivers
 - Manages specific hardware devices
 - In an OS, only drivers are device-specific
 - Have knowledge about its own device
 - OS provides a general **driver interface** to allow easy cooperation of OS and its drivers
 - **Memory management** of I/O including
 - **buffering** (storing data temporarily while it is being transferred)
 - **caching** (storing parts of data in faster storage for performance)
- Description here is also valid for storage devices
 - Disks are IO devices

Protection and Security



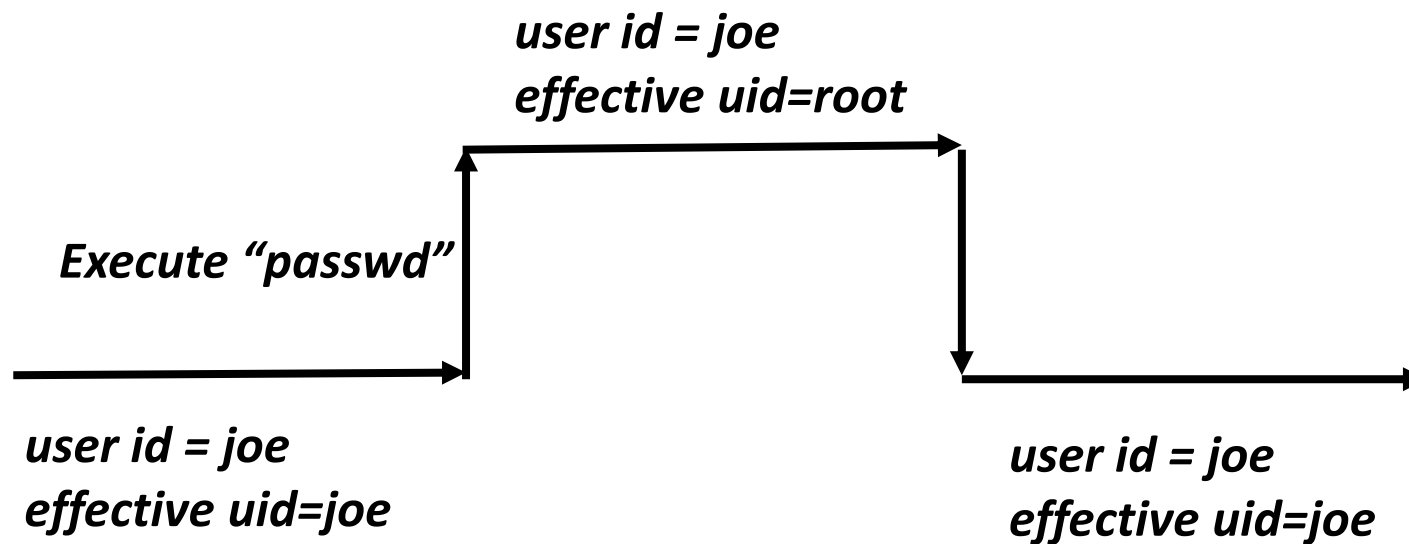
- **Protection** – mechanisms for controlling **access** (of processes or users) **to resources**
 - Distinguish between authorized and unauthorized usage
- **Security** – defense of the system against internal and external attacks
 - Huge range, including denial-of-service, viruses...

Protection and Security

- Systems generally first distinguish among **users**, to determine who can do what
 - Each user has a user identifier (**user ID**)
 - User IDs are associated with all files & processes to determine access control
 - **Process owners** and **file owners** are all users...
 - Group identifier (**group ID**) allows set of users to be defined and managed, also associated with each process or file
 - **Privilege escalation** allows user to change to effective ID with more rights *temporarily*
 - **Setuid** in UNIX (see next slide)
 - Causes a program to run with a **user ID of the file owner**, rather than the user's ID
 - » E.g., change your own password, which requires the root privilege to update the password file.
 - Might become a vulnerability if the program is attacked...

Setuid in Linux

```
cissoll> ls -al passwd  
-r-sr-sr-x 1 root sys 27228 Aug 17 2007 passwd*  
cissoll>
```

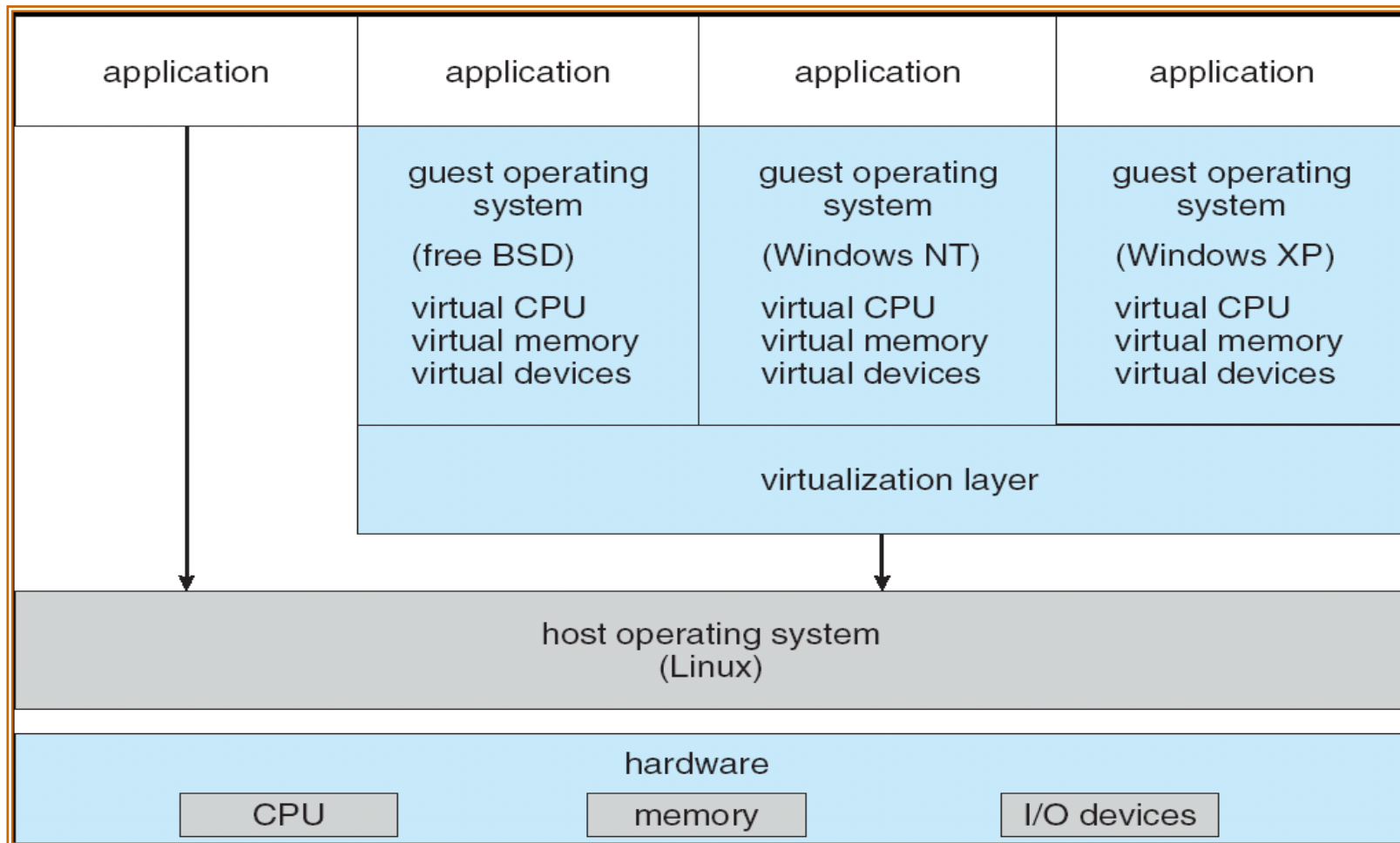


Virtualization

- **Virtualization**

- Abstract **the hardware of a single computer** into several different execution environments
- Create an illusion that each environment is running on its own private computer
 - An OS can be run in each environment
- Allows OSes to run on other OSes

Virtualization



host operating system vs. *guest* operating system

Virtualization

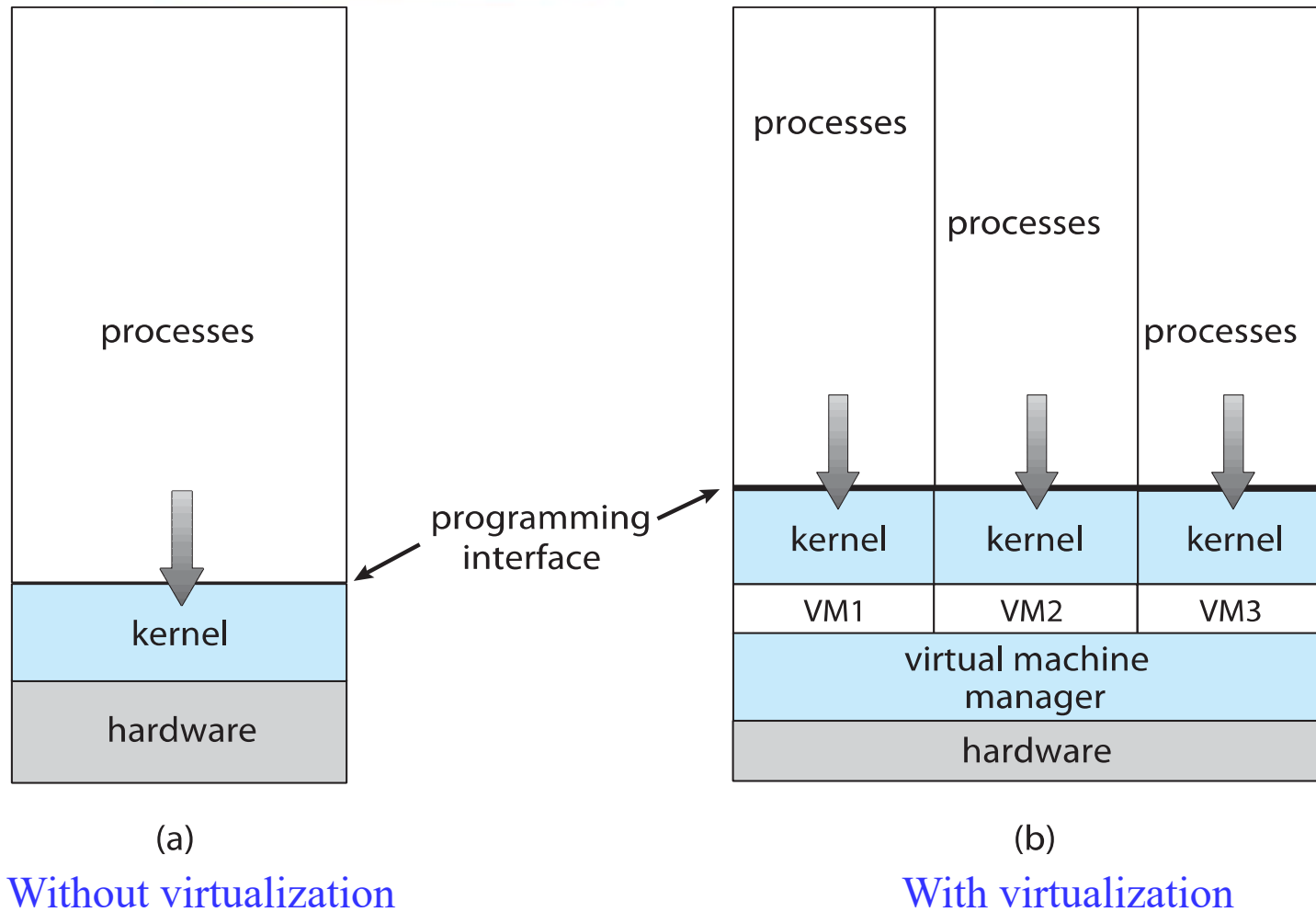
- **Emulation** –

- Simulating computer hardware in software
 - Used when source CPU type **different** from target type
- For Example
 - We want to run the PowerPC applications on the Intel x86
 - Every instruction of PowerPC must be **translated** to the instructions on Intel x86
- Slow...

Virtualization (Cont.)

- **Virtualization** –
 - An OS **natively compiled for a CPU**, running on another OS **also natively compiled to that CPU**
 - Example
 - Run Linux for x86 on the Windows 10 (also for x86)
 - Linux: **guest OS**
 - Windows 10: **host OS**
 - **VMM (Virtual Machine Manager)** provides virtualization services
 - Also called **Hypervisor**
 - VMM can also run **natively** without the host OS
 - Example: VMware ESX and Citrix XenServer

VMM without Host OS



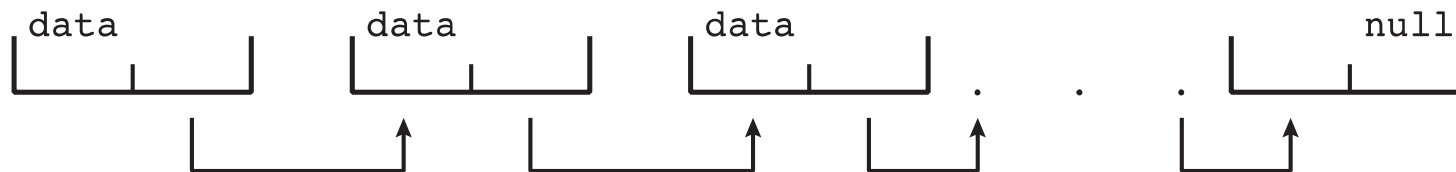
VMM directly runs on hardware (do not need host OS)

Kernel Data Structures

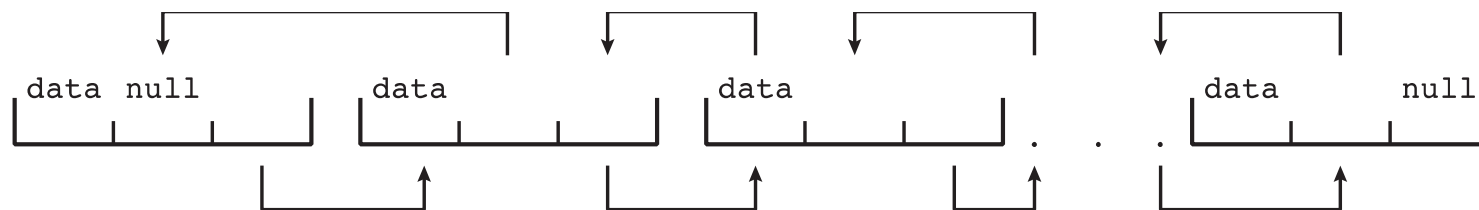
- Lists, Stacks, and Queues
- Trees
- Hash Tables
- Linux data structures defined in *include* files
 - e.g., `<linux/list.h>`, `<linux/kfifo.h>`,
`<linux/rbtree.h>`

List

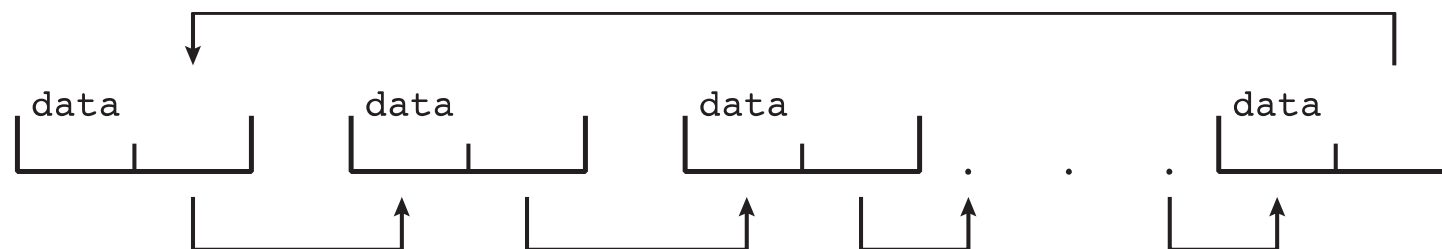
- *Singly linked list*



- *Doubly linked list*



- *Circular linked list*



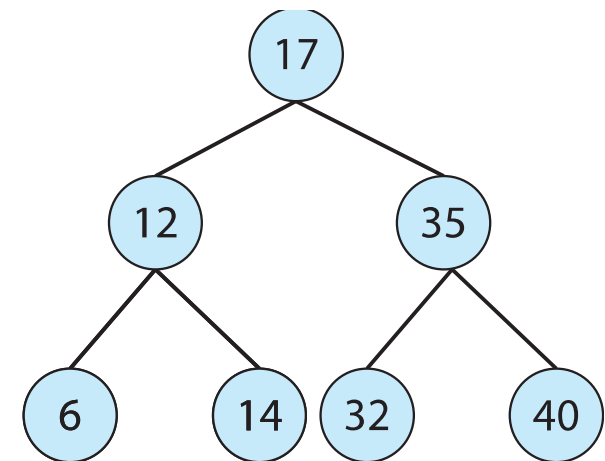
Stack and Queue



- **Stack:** last in, first out (LIFO)
 - Insert an item: **push**
 - Remove an item: **pop**
- **Queue:** first in, first out (FIFO)

Tree

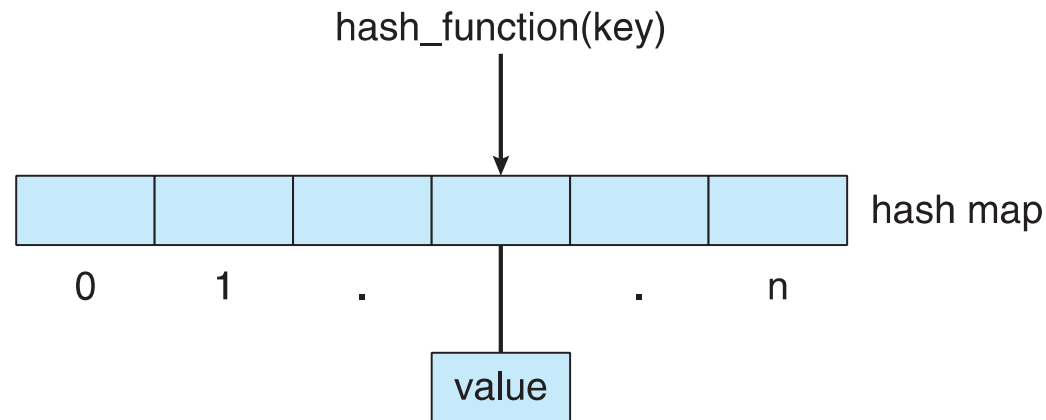
- **General tree**
 - A parent may have an unlimited number of children
- **Binary tree**
 - A parent may have at most two children
- **Binary search tree**
 - Two children, left \leq right
 - Search performance is $O(n)$
 - **Balanced binary search tree**
 - Search performance is $O(\lg n)$



Binary search tree

Hashing and Bitmap

- **Hash function** can create a **hash map**



- **Bitmap** – string of n binary digits representing the status of n items

Computing Environments



- Traditional Computing
- Mobile Computing
- Client-Server Computing
- Peer-to-Peer Computing
- Cloud Computing
- Real-Time Embedded Systems

Computing Environments



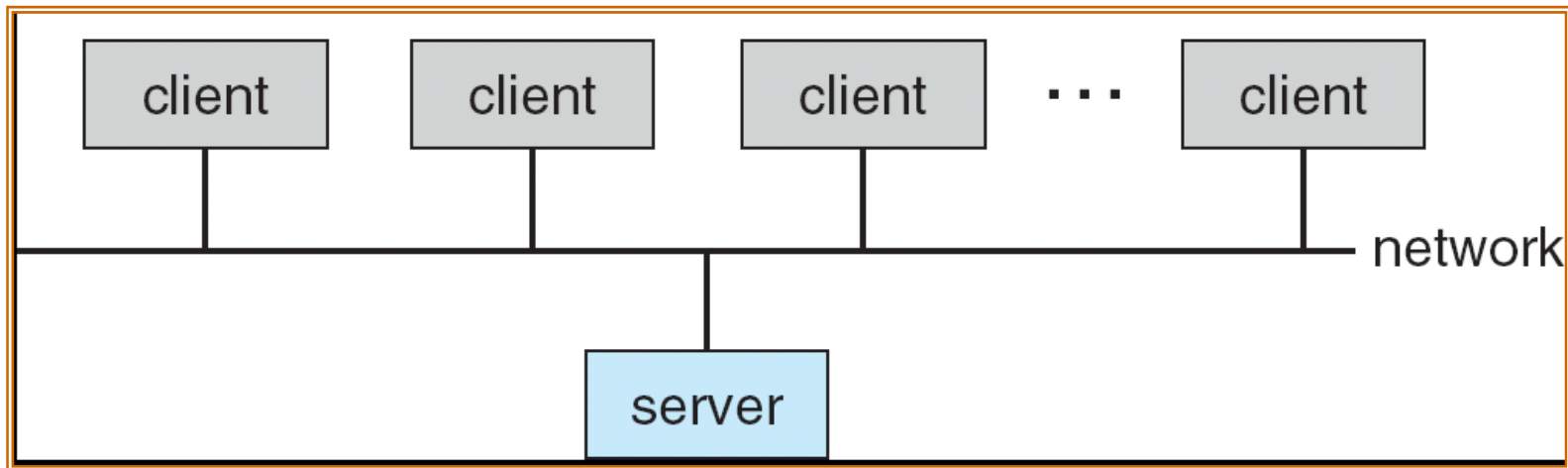
- Traditional Computing
 - Office environment
 - PCs connected to a network, **terminals** attached to **mainframe or minicomputers** providing **batch and timesharing** functionalities
 - Now **portals** allowing remote systems to access internal resources
 - Home networks
 - Used to be single system, then modems
 - Now firewalled, networked

Mobile Computing

- Computing on **handheld devices**
 - Smartphones
 - Tablets
- Issues
 - Limited memory
 - Slower processors
 - I/O constraints: small display screens, small keyboards.....
 - Limited power
- Extra features
 - GPS, accelerometers, and gyroscope
- Leaders are **Apple iOS** and **Google Android**

Client-Server Computing

- **Servers**: serving requests, **Clients**: issuing requests
 - ▶ **Application-server** provides an interface to client to request services
 - ▶ **File-server** provides interface for clients to store and retrieve files
- For clients, dumb terminals are replaced by PCs or smart phones

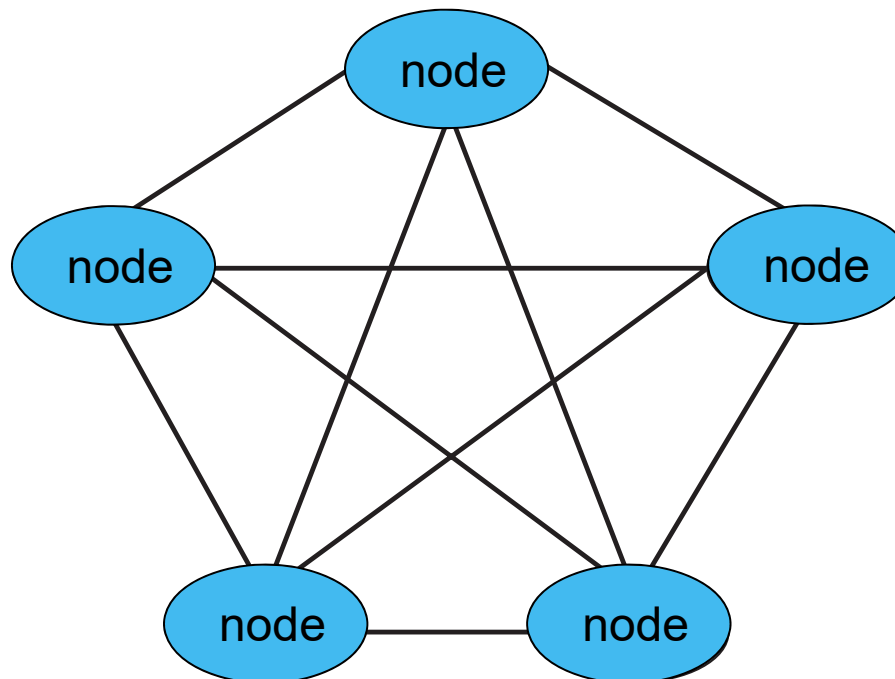


Peer-to-Peer Computing

- P2P does not distinguish clients and servers
 - Instead all nodes are considered *peers*
 - May each act as client, server or both
 - Node must join P2P network
- Advantage
 - In client-server system, the server may become a bottleneck.
In P2P, services can be provided by several nodes
- Example
 - skype

Peer-to-Peer Computing

- Node must join P2P network
 - Registers its service(s) with central lookup service on network, or
 - Broadcast requests for services and respond to requests for services via *discovery protocol*

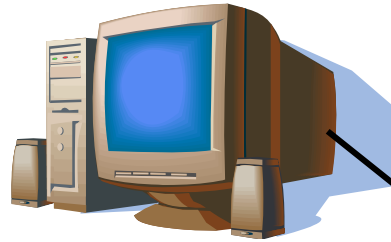


Cloud Computing

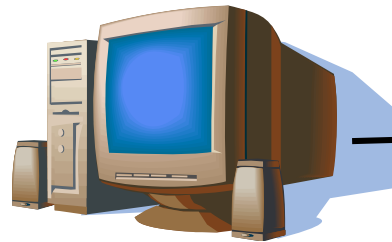
- Delivers computing, storage, even apps as a service across a network
- Logical extension of **virtualization** (see next slide)
 - Typically based on virtualization
 - Amazon **EC2** has thousands of servers, millions of **VMs**, PBs of storage available across the Internet, pay based on usage
- Different types
 - **Public cloud** – available via Internet to anyone willing to pay
 - **Private cloud** – run by a company for the company's own use
 - **Hybrid cloud** – includes both public and private cloud components

Logical Extension of Virtualization

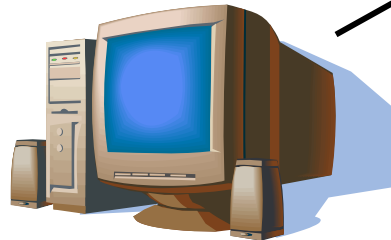
WWW Server
Solaris



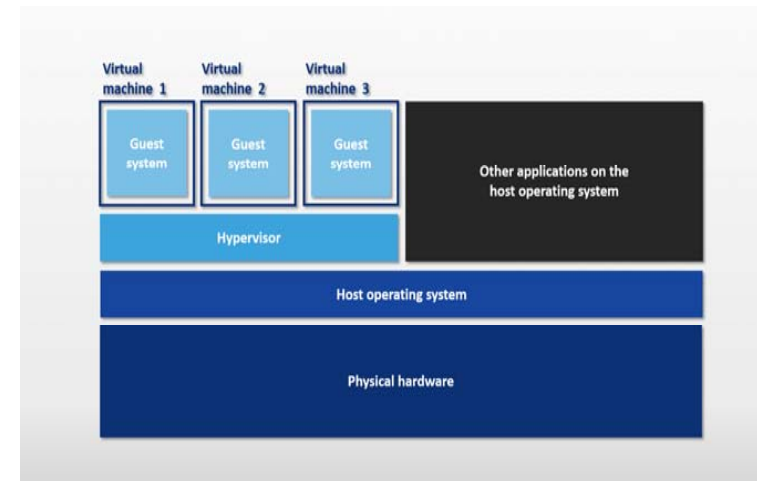
File Server
Win10



Database Server
Linux



**WWW + File + Database
Servers**

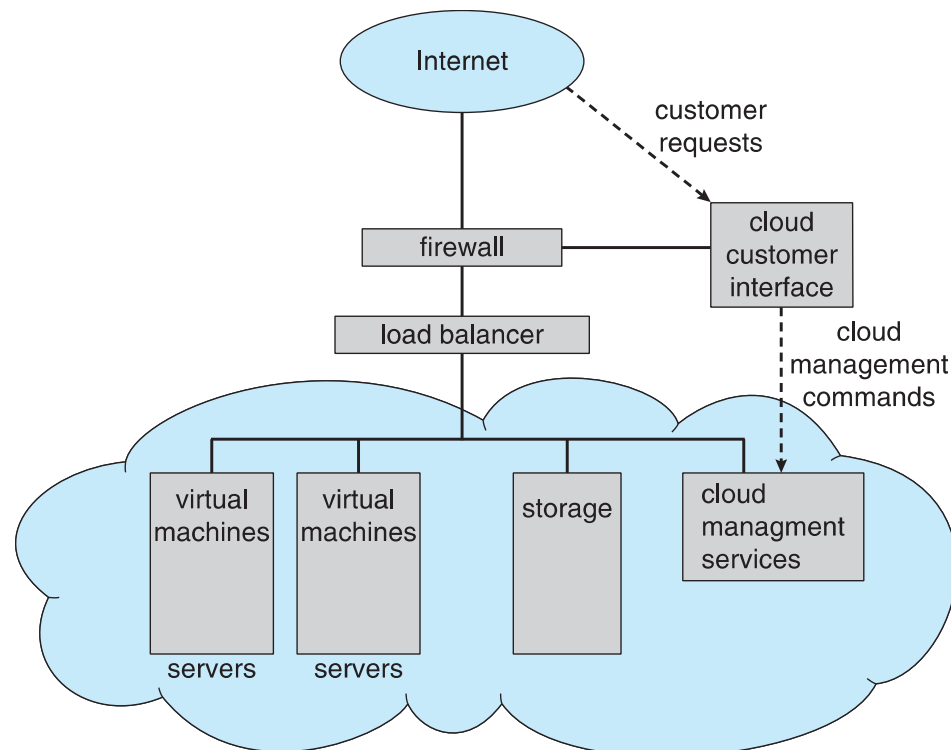


Cloud Computing

- Categories
 - **Software as a Service (SaaS)** –
 - one or more applications available via cloud (e.g., word processor)
 - **Platform as a Service (PaaS)** –
 - software stack ready for application use via cloud (e.g., a database server)
 - **Infrastructure as a Service (IaaS)** –
 - servers or storage available via cloud (e.g., storage available for backup use)

Cloud Computing

- Cloud computing environments composed of traditional OSES, plus VMMs, plus cloud management tools
 - Internet connectivity requires security like firewalls
 - Load balancers spread traffic across multiple applications



Real-Time Embedded Systems



- Embedded computers
 - Most prevalent form of computers
 - Cars, routers, robots, printers,...
 - Tend to have very **specific** tasks
- Different OS implementation schemes
 - No OS
 - A tiny embedded OS
 - A standard OS (e.g., Linux)

Real-Time Embedded Systems

- Embedded systems sometimes run *real-time operating systems*
 - Processing must be done within a predefined time constraint (i.e., **deadlines**)
 - Otherwise, the system will fail
 - e.g., missile systems

Open-Source Operating Systems



- Many operating systems are open source
 - e.g., Linux, FreeBSD, OpenSolaris, Illumos,...